Are Transactions Right For You?
Using and Not Abusing Transactions

Sheeri Cabral
Product Manager, MongoDB
We can’t cross the tracks. The train comes! That would be a big problem.

Because….’cause we waste yucky gas.
We use transactions to avoid having a big problem.

Using unnecessary transactions is a waste of resources.
MongoDB Transactions

MongoDB has transactions

More than one document

More than one shard
What is a transaction?

“A transaction is a data state transition”
- Vlad Mihalcea

_A Beginner’s Guide to ACID and Database Transactions_

Data State Transition

Before a transaction
Data is in one state

After a transaction
Data is in another state

Canonical example
Bank transfer
ACID Transactions

...and the problems they solve
Atomicity

All actions either succeed or fail

There is no partial success
Atomicity

All actions either succeed or fail
There is no partial success

Data state before
$1000 in savings, $500 in checking

Data state after
$900 in savings, $600 in checking

Total: $1500
Atomicity

Total: $1600

Total: $1500

Total: $1400
Atomicity

All actions either succeed or fail
There is no partial success

Data state before
$1000 in savings, $500 in checking

Data state after
$900 in savings, $600 in checking
Consistency

Different from CAP Theorem
ACID consistency is not related to data freshness

Follow the Rules
“any given database transaction must change affected data only in allowed ways”

Data Correctness
Data types
Key constraints
Data validation
Triggers
Isolation

Levels
Can a transaction see writes inside another concurrent transaction?

Tradeoff
Performance vs. accuracy

Issues
Dirty reads
Non-repeatable reads
Phantom reads
**Durability**

- **No losing data**
  - New state is permanent

- **Hard to achieve**
  - Log, then write

- **Recovery**
  - Looks at log
Why We Use ACID Transactions

Atomicity
Everything happens or nothing happens

Consistency
Data follows the rules

Isolation
Controls what is seen mid-transaction by other transactions

Durability
Changes are saved
<table>
<thead>
<tr>
<th>Summer 2018</th>
</tr>
</thead>
<tbody>
<tr>
<td>Took 9 years (1&lt;sup&gt;st&lt;/sup&gt; release 2009)</td>
</tr>
<tr>
<td>Document model</td>
</tr>
<tr>
<td>Transactions are hard!</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MySQL</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Innobase</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fully merged in 2013</td>
</tr>
</tbody>
</table>
Relational DBs and MongoDB: Transactions are slower
Do you really need a transaction?

Social media: dirty/phantom reads are OK.

Airplane ticketing and seating

US restaurants

ebay doesn’t use transactions.


“The rationale for not using transactions was that they harm performance at the sort of scale that eBay deals with.”
Relational DBs and MongoDB:

Transactions are slower
Going Transactionless in the Database

Code-your-own transactions in the app layer

Optimistic concurrency control – online shopping cart

Pessimistic concurrency control – ticket sales and timers

Controllable performance hit
Use fewer transactions
Use Fewer Transactions

Which steps need transactions?

Ordering a coffee:

- order/pay  
  Strictly ordered queue

- making/receiving coffee  
  Not-so-strictly ordered queue

Get transactional behavior without transactions
Relational Model

Atomicity

<table>
<thead>
<tr>
<th>savings</th>
<th>account_id</th>
<th>balance</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>111</td>
<td>900</td>
</tr>
</tbody>
</table>

UPDATE savings
SET balance=balance-100
WHERE account_id=111

UPDATE checking
SET balance=balance+100
WHERE account_id=222

<table>
<thead>
<tr>
<th>checking</th>
<th>account_id</th>
<th>balance</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>222</td>
<td>600</td>
</tr>
</tbody>
</table>
Accounts

<table>
<thead>
<tr>
<th>account_id</th>
<th>balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>111</td>
<td>900</td>
</tr>
<tr>
<td>222</td>
<td>600</td>
</tr>
</tbody>
</table>

UPDATE savings
SET balance=balance-100
WHERE account_id=111

UPDATE checking
SET balance=balance+100
WHERE account_id=222

Relational Model

Atomicity
{  
"_id" : ObjectId("5ea7137e2295dc3b6afb6057"),  
"name" : "Sheeri Cabral",  
"accounts" : [  
{ "id" : 111,  
  "type" : "savings",  
  "balance" : NumberDecimal("1000") },  
{ "id" : 222,  
  "type" : "checking",  
  "balance" : NumberDecimal("500") }  
],  
"phone" : "+1 123-456-7890",  
"email" : "sheeri.cabral@mongodb.com" 
}
People

{
  "_id" : ObjectId("5ea7137e2295dc3b6afb6057"),
  "name" : "Sheeri Cabral",
  "phone" : "+1 123-456-7890",
  "email" : "sheeri.cabral@mongodb.com"
  "account_ids" : [ 111, 222 ]
}

Accounts

{
  "id" : 111,
  "type" : "savings",
  "balance" : NumberDecimal("1000")
}

{
  "id" : 222,
  "type" : "checking",
  "balance" : NumberDecimal("500")
}
Sheeri Cabral

accounts: [ 
{ "id" : 111,
  "type" : "savings",
  "balance" : NumberDecimal("1000") },
{ "id" : 222,
  "type" : "checking",
  "balance" : NumberDecimal("500") }
],

phone: "+1 123-456-7890",
email: "sheeri.cabral@mongodb.com"

```
db.getCollection("accounts").updateOne(
  { "_id": ObjectId("5ea7137e2295dc3b6afb6057") },
  { $inc: {
    "accounts.$[sv].balance": -100,
    "accounts.$[ck].balance": 100
  } },
  { arrayFilters: [
    { "sv.id": 111 },
    { "ck.id": 222 }
  ]
});
```
Relational DBs offer **transactional behavior** without transactions in **ONE** case: changing one row
MongoDB has more options for transactional behavior without transactions
Document Model

{ 
  "_id" : ObjectId("7137e2295dc3b6af60575ea"),
  "name" : "Antonio Cabral",
  "accounts" : [
    { "id" : 333,
      "type" : "checking",
      "balance" : NumberDecimal("700") }
  ],
}

{ 
  "_id" : ObjectId("5ea7137e2295dc6afb60575ea"),
  "name" : "Sheeri Cabral",
  "accounts" : [
    { "id" : 111,
      "type" : "savings",
      "balance" : NumberDecimal("1000") },
    { "id" : 222,
      "type" : "checking",
      "balance" : NumberDecimal("500") },
  ],
  "phone" : "+1 123-456-7890",
  "email" : "sheeri.cabral@mongodb.com"
}
MongoDB has transactional behavior without transactions if you model your data that way.
MongoDB has transactional behavior with fewer transactions if you model your data that way.
Transactions in MongoDB are multi-document
MongoDB Consistency / Data Rules

Flexible schema != lack of schema

Similar to relational world:

• Unique indexes, _id
• Required fields, data type
• Enum
MongoDB Consistency / Data Rules

• More flexibility with MongoDB

• Range

• Most query selectors, including $regex and complex logic

• validationLevel
  • strict (all)
  • moderate (grandfathered in)

• validationAction
  • error (reject)
  • warn (accept)
Isolation

Issues
- dirty reads
- non-repeatable reads
- phantom reads
- none of the above

What should happen?
Tradeoff
MongoDB
Isolation
readConcern OUTSIDE of a transaction
readConcern

vs.

readPreference
readConcern OUTSIDE of a transaction

majority Read Committed
MongoDB

Isolation

readConcern OUTSIDE of a transaction
- majority: Read Committed
- linearizable: Read Committed
- local
readConcern OUTSIDE of a transaction

- majority: Read Committed
- linearizable: Read Committed
- local: Read Uncommitted

MongoDB Isolation
readConcern OUTSIDE of a transaction

- majority: Read Committed
- linearizable: Read Committed
- local: Read Uncommitted
- available: Read Uncommitted

MongoDB Isolation
readConcern INSIDE a transaction
- ONE readConcern

<table>
<thead>
<tr>
<th>isolation</th>
<th>readConcern</th>
</tr>
</thead>
<tbody>
<tr>
<td>majority</td>
<td>Read Committed</td>
</tr>
<tr>
<td>linearizable</td>
<td>(not available)</td>
</tr>
<tr>
<td>local</td>
<td>Read Uncommitted</td>
</tr>
<tr>
<td>available</td>
<td>(not available)</td>
</tr>
<tr>
<td>snapshot</td>
<td>Repeatable Read</td>
</tr>
<tr>
<td>multi-doc txn</td>
<td></td>
</tr>
</tbody>
</table>

MongoDB Isolation
MongoDB Durability

writeConcern

Within transactions or not

Outside of transactions, per-operation level
Per-transaction level for multi-document transactions

<number>
  • 0
  • 1

“majority”  <---- This makes MongoDB durable, without transactions!
ACID Transactions in the Relational World

Atomicity
Everything happens or nothing happens

Consistency
Schema validation – extremely flexible

Isolation
Controls what is seen mid-transaction by other transactions

Durability
Changes are saved

MongoDB Outside of Transactions

Atomicity
Multiple writes in one document are atomic

Consistency
Schema validation – extremely flexible

Isolation
readConcern - READ COMMITTED
        READ UNCOMMITTED

Durability
writeConcern
MongoDB Transactions

**Atomicity**
Multi-document, multi-shard atomic writes

**Consistency**
Schema validation – extremely flexible

**Isolation**
- readConcern: READ COMMITTED
  - REPEATABLE READ: READ UNCOMMITTED

**Durability**
writeConcern

---

MongoDB Outside of Transactions

**Atomicity**
Multiple writes in one document are atomic

**Consistency**
Schema validation – extremely flexible

**Isolation**
- readConcern: READ COMMITTED
  - REPEATABLE READ: READ UNCOMMITTED

**Durability**
writeConcern
Using MongoDB Transactions

function transferMoney( dbName, session, src, trg, amount ) {
    session.start_transaction()
    const db = session.get_database(dbName)

    db.accounts.updateOne(
        { $and: [{"_id": src.uid}, { "accounts.id": src.acctId } ] },
        { $inc: { "accounts.$.balance" : -amount } } )

    db.accounts.updateOne(
        { $and: [{"_id": trg.uid}, { "accounts.id": trg.acctId } ] },
        { $inc: { "accounts.$.balance" : amount } } )

    session.commit_transaction()
}
MongoDB Transaction Limitations

Time: 60s

Number of documents: no limit

Transient failures

- catch and re-try transaction
- use retryable writes

Existing transactions block DDL

Existing DDL blocks transactions
MongoDB Transaction Limitations - sharding

More shards = more resources used

Shards with arbiters will abort transactions

Transactions have an impact on collections that do chunk migrations
Thank you

A Beginner’s Guide to ACID and Database Transactions by Vlad Mihalcea

Wikipedia entry on consistency

Martin Fowler on ebay not using transactions

Further topic:

SELECT...for UPDATE inside MongoDB Transactions

Sheeri Cabral | Product Manager | MongoDB | @sheeri